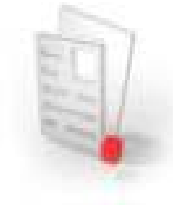
California Enterprise Architecture Program



SOA *White Paper*

SOA Security

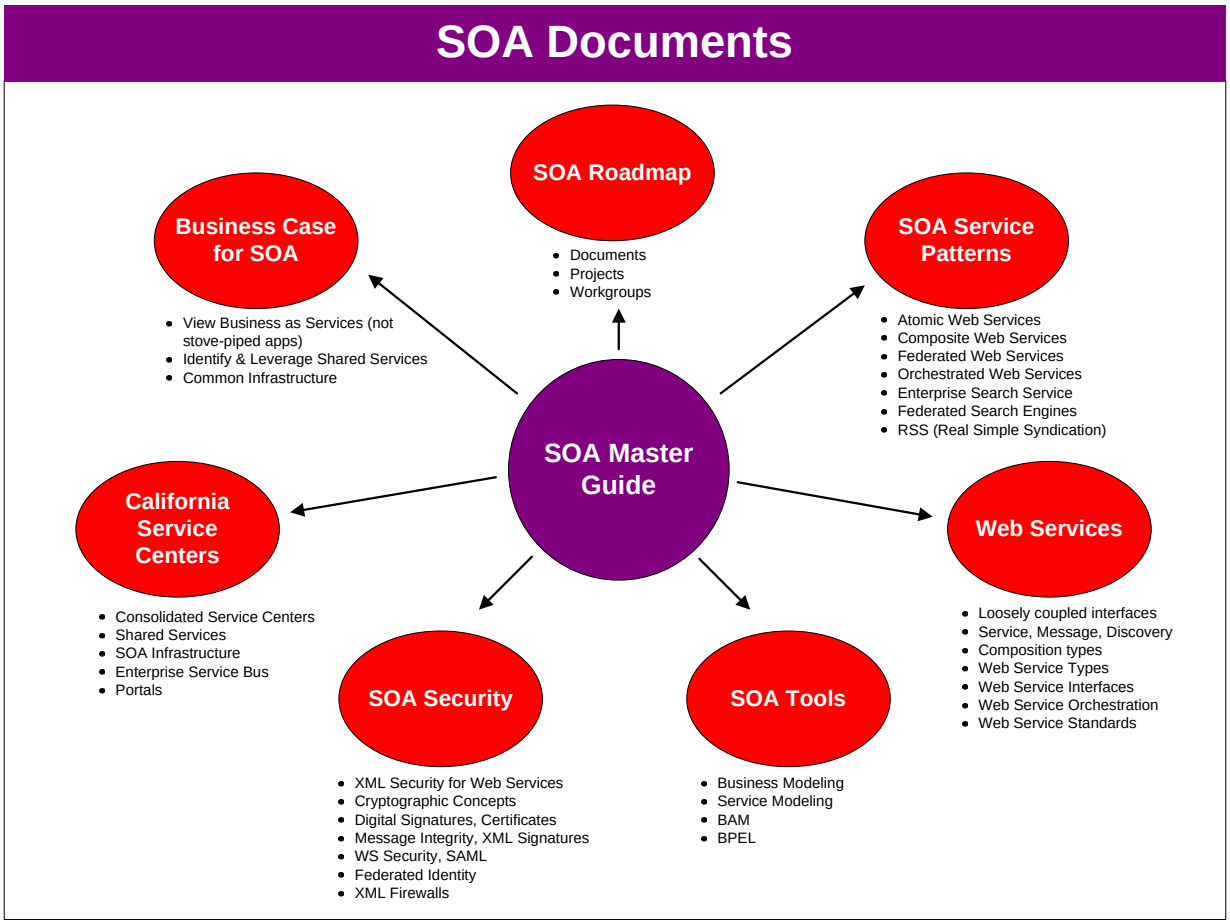
Revision History

06/30/2006	Original Draft
08/02/2006	Added new sections to Web Services Security and Identity & Authentication
09/11/2006	Revised Federated Identity section. Added placeholder for User Provisioning.

Table of Contents

SOA Documents

The service oriented architecture advocated by the California Enterprise Architecture Program is organized into a set of interrelated documents. A master guide serves as the "jumping off point" and describes in an overview fashion the key parts of SOA.



There are six white papers planned to address in depth details of SOA. This whitepaper is *SOA Security*.

In essence, this is *Web Single Sign On*:

A security token is presented to a gatekeeper in order for a user to get authenticated. Now imagine that the gatekeeper is guarding the main gate of a large building with many offices. Visitors are required to show their ID cards and get authenticated at the main gate. The gatekeeper checks the ID card by matching it with his internal record and then allows the visitor to enter the building.

Let's suppose that you want to visit several offices in the building. Each office has an entrance with a gatekeeper guarding the entrance of each office. You need to get authenticated at the entrance of each office. The gatekeeper at the entrance of each office repeats the same authentication act.

What if individual offices in the building trust the authentication performed by the gatekeeper of the main gate? The building will become a trusted domain, of which each individual office will be a part. Naturally if this type of trust exists between different offices, they would like to share the processing load of the authentication act.

A possible solution to allow sharing of authentication information is to issue a temporary identification badge to a visitor at the main gate of the building. The gatekeeper at the main gate will issue a badge to each visitor after successful authentication. The identification badge will have a short expiry. The visitor will show the identification badge while entering each office. The office gatekeeper will check the validity of badge before allowing or disallowing a person to enter the office.

So, the "gatekeeper" is the *Identity Authority* for this office building, which, lets say, handles one type of *Citizen* business (perhaps vehicle registrations). Now, let's assume that we have a second office building that handles a different type of citizen business (perhaps professional licensing) that also has its own gatekeeper. If the two office buildings agree to "trust" the first gatekeeper, then we have a "federated identity" relationship. That is, one of the gatekeeper's is designated as the Identity Authority (in this case, for citizens) and the other gatekeepers trust the first gatekeeper and allows entrance to their respective buildings without re-authenticating the visitor.

Further, let's expand our scenario and say we have an additional office building for *Employees*, and another building for *Business* (to business) activities. Each office building could have a single "gatekeeper", for example an Employee Authority and a Business Authority, respectively. But, what if a visitor is an agent that represents a business but is also a citizen and conducts both types of business? This complicates things and probably the only reasonable solution is the visitor must be authenticated based on the type of business they wish to transact at that point in time. If it is citizen business, then the visitor must go to the Citizen Authority. If acting as a business agent, then the visitor must be authenticated by the Business Authority. However, once authenticated by the Citizen Authority, the visitor can interact with *any* building office that deals with citizens – as long as the office buildings are in the *same type of trusted relationship*. See Identity Authorities Pattern and Communities of Interest Pattern in this paper for more details.

Such scenarios are common in Enterprise Application Integration. Whether applications are running within or across the boundaries of an enterprise, the sharing of authentication information forms an important part of application integration effort. Naturally, the sharing of authentication information prevents each application from having to perform the entire authentication process.

In the new SOA-based environment, a number of business services will contain some sort of sensitive information and therefore, will require some level of security. Fortunately, there are a lot of choices defined in the Security Standards for Web Services.

Since web services interoperate via XML documents (SOAP messages), the standards provide for security details within the XML definitions. The overall XML structures for security are defined in the WS-Security standard.

Introduction

Exchange of information over the Internet is vital but may have security implications. Security issues over the Internet are important, because it is an insecure and untrustworthy public network infrastructure, prone to attacks by intruders.

The information available over the Internet does not always have the same level of business confidentiality. In the public sector, much information is intended to be accessed and viewed by anyone. However, there are a number of business transactions that require knowing who the party is including the party's access privileges.

Organizations usually secure company resources available on the network and online services by defining business roles, access rights, and system policies. That's where firewalls play a role in the security process. A network level firewall sits at the doorstep of a private network as a guard and typically provides the following security services:

- monitors all incoming traffic;
- checks the identity of requesters trying to access specific company resources;
- authenticates user identities, which can be the network addresses of service requesters or security tokens;
- checks security and business policies to filter access requests and verifies whether the service requester has the right to access the intended resource; and
- provides for encrypted messages so that confidential information can be sent across the Internet.

The main purpose of a firewall is to protect the physical boundaries of a network. There is a physical boundary of the private network and the only way to get into the network is through the firewall. While packets of network traffic and messages pass through a firewall, they are authenticated and checked for intrusion or malicious attacks.

When a department application interacts with an enterprise component provided by a different department, it cannot control and may not even know much about the IT infrastructure of the other department. For example, the first department might be using a Java solution over Solaris servers and the second department may be using .NET or some other technology. How can interoperability between departments be ensured?

It is not feasible to sit down with all departments and decide about messaging formats for exchange of information and interoperability. This will create an endless task of designing and redesigning message formats for each department. The cost of this type of legacy integration is so high that such techniques are only feasible in high IT return private sectors, such as airline and banking industries. Most government e-commerce isn't able to justify IT infrastructure development costs in this way.

But if applications shift from legacy integration to the Service Oriented Architecture (SOA) provided by web services, interoperability issues are much easier to deal with. A web services-based SOA depends on SOAP servers to process messages. A SOAP server holds only the information related to the web services it is hosting (names of the services, names of the methods in each service, where to find the actual classes that implement the web services, and so on) and has the capability of processing incoming SOAP requests. However, the SOAP server itself doesn't have any capability to check whether the incoming SOAP request is coming from an anonymous customer or a known business partner. SOAP

cannot distinguish between sensitive and non-sensitive web services and cannot perform user authentication, authorization, and access control.

It is clear that a remote client who has accessed a SOAP server enjoys the opportunity of invoking any method of any services hosted on the particular SOAP server. So, one might correctly conclude that it is not safe to host web services of different levels of sensitivity on the same SOAP server.

Even if you deploy a network-level firewall to protect from intruders, you will not be able to distinguish between different users once it has reached the SOAP server. It is possible that an intruder authenticates himself as an anonymous user, reaches the SOAP server, and invokes sensitive web services meant for a different user. Thus, a SOAP server is like a hole in your network.

There are two solutions to this problem:

1. Use a different SOAP server for each level of sensitivity, so that different authentication policies can be enforced on each sensitivity level. This solution may seem appropriate for web services today. However the real advantage of web services lies in the next generations where web services will not just be invoked by browser-assisted human clients, but web services will invoke each other to form chained or transactional operations. Such complex web service infrastructure will be very hard and expensive to build with the idea of having a separate SOAP server for each authorization policy. In addition, this idea hardly allows building reusable or off-the-shelf security solutions.

2. The second option is to make the firewall XML and SOAP aware. The firewall will be able to inspect SOAP messages, trying to match user roles with access lists, policy levels, and so on. This solution is a better approach. It also allows building XML-based standard security protocols, which can be adopted by security vendors to ensure interoperability.

Web service users can add security information (signature, security tokens, and algorithm names) inside SOAP messages, according to the XML-based security protocols. The XML and SOAP-aware firewall will check the message before it reaches the SOAP server, so that it is able to detect and stop intruders before they are able to reach the service.

Based on the second approach described above, W3C and OASIS are developing several XML-based security protocols. These protocols will define the various security features of an XML and SOAP-aware firewall.

XML Security for Web Services

This section briefly discusses some of the high level features of security protocols from W3C and OASIS.

The *XML Signature* specification is a joint effort of W3C and IETF. It aims to provide data integrity and authentication (both message and signer authentication) features, wrapped inside XML format.

W3C's *XML Encryption* specification addresses the issue of data confidentiality using encryption techniques. Encrypted data is wrapped inside XML tags defined by the XML Encryption specification.

WS-Security from OASIS defines the mechanism for including integrity, confidentiality, and single message authentication features within a SOAP message. WS-Security makes use of the XML Signature and XML Encryption specifications and defines how to include digital signatures, message digests, and encrypted data in a SOAP message.

Security Assertion Markup Language (SAML) from OASIS provides a means for partner applications to share user authentication and authorization information. This is essentially the single sign-on (SSO) feature being offered by all major vendors in their e-commerce products. In the absence of any standard protocol on sharing authentication information, vendors normally use cookies in HTTP communication to implement SSO. With the advent of SAML, this same data can be wrapped inside XML in a standard way, so that cookies are not needed and interoperable SSO can be achieved.

eXtensible Access Control Markup Language (XACML) presented by OASIS lets you express your authorization and access policies in XML. XACML defines a vocabulary to specify subjects, rights, objects, and conditions -- the essential bits of all authorization policies in today's e-commerce applications.

Basic Cryptographic Concepts

The discussion of message integrity, user authentication, and confidentiality employs some core concepts: keys, cryptography, signatures, and certificates. Following, cryptographic basics will be briefly discussed.

Asymmetric cryptography

A popular cryptographic technique is to use a pair of keys consisting of a public and a private key. First, you use a suitable cryptographic algorithm to generate your public-private key pair. Your public key will be open for use by anyone who wishes to securely communicate with you. You keep your private key confidential and do not give it to anybody. The public key is used to encrypt messages, while the matching private key is used to decrypt them.

In order to send you a confidential message, a person may ask for your public key. He encrypts the message using your public key and sends the encrypted message to you. You use your private key to decrypt the message. No one else will be able to decrypt the message, provided you have kept your private key confidential. This is known as *asymmetric encryption*. Public-private key pairs are also sometimes known as *asymmetric keys*.

Symmetric cryptography

There is another encryption method known as symmetric encryption. In symmetric encryption, you use the same key for encryption and decryption. In this case, the key has to be a shared secret between communication parties. The shared secret is referred to as a symmetric key. Symmetric encryption is computationally less expensive than asymmetric encryption. This is why asymmetric encryption is ordinarily only used to exchange the shared secret. Once both parties know the shared secret, they can use symmetric encryption.

Message digests

Message digests are another concept used in secure communications over the Internet. Digest algorithms are like hashing functions: they consume (digest) data to calculate a hash value, called a message digest. The message digest depends upon the data as well as the digest algorithm. The digest value can be used to verify the integrity of a message; that is, to ensure that the data has not been altered while on its way from the sender to the receiver. The sender sends the message digest value with the message. On receipt of the message, the recipient repeats the digest calculation. If the message has been altered, the digest value will not match and the alteration will be detected.

But what if both the message and its digest value are altered? That kind of change may not be detectable at the recipient end. So a message digest algorithm alone is not enough to ensure message integrity. That's where we need digital signatures.

Digital signatures

Keys are also used to produce and verify digital signatures. You can use a digest algorithm to calculate the digest value of your message and then use your private key to produce a digital signature over the digest value. The recipient of the message first checks the integrity of the hash value by repeating the digest calculation. The recipient then uses your public key to verify the signature. If the digest value has been altered, the signature will not verify at the recipient end. If both the digest value and signature verification steps succeed, you can conclude the following two things:

- The message has not been altered after digest calculation (message integrity); and
- The message is really coming from the owner of the public key (user authentication).

Certificates

In its most basic form a digital certificate is a data structure that holds two bits of information:

1. The identification (e.g. name, contact address, etc.) of the certificate owner (a person or an organization).
2. The public key of the certificate owner.

A certificate issuing authority issues certificates to people or organizations. The certificate includes the two essential bits of information, the owner's identity and public key. The certificate issuing authority will also sign the certificate using its own private key; any party can verify the integrity of the certificate by verifying the signature.

Message Integrity and User Authentication with XML Signatures

The XML Signature specification, XML Digital Signature, (XMLDS) has been jointly developed by W3C and IETF. It has been released as a recommendation by W3C. XML Signature defines the

processing rules and syntax to wrap message integrity, message authentication, and user authentication data inside an XML format.

As an example, a department includes message integrity and user authentication information within a SOAP method invocation. The XML firewall of the department receiving the message, on receipt of the invocation, will need to look into the SOAP message to verify that:

- The message has not been altered while on its way to the web service (message integrity); and
- The requester is really the trusted user (user authentication).

The XML firewall will only let the request pass onto the SOAP server if both these conditions are met. It should be noted that XMLDS, isn't SOAP-specific. XMLDS can be used to insert signatures and message digests into any XML instance, SOAP or otherwise.

An XMLDS implementation can create SOAP headers to produce signed SOAP messages. The XML firewall sitting at the recipient's end will process the SOAP header to verify the signatures before forwarding the request to the SOAP server. We can achieve the following two security objectives through this procedure:

- We can verify that the SOAP message that we received was really sent by the sender we think it came from.
- We can verify that the data we received has not been changed while on its way and is the same that the sender intended to send.

XML Encryption

The XML Encryption specification satisfies confidentiality requirements in XML messages. XML encryption offers several features.

- You can encrypt a complete XML file.
- You can encrypt any single element of an XML file.
- You can encrypt only the contents of an XML element.
- You can encrypt non-XML data (e.g. a JPG image).
- You can encrypt an already encrypted element (i.e., "super-encryption").

XML Encryption Processing

How will our XML firewall work with these encryption concepts? It will receive SOAP messages with encrypted elements or content and translate the contents to a decrypted form before forwarding the decrypted SOAP message request to the SOAP server.

The recipient of an XML encrypted file will decrypt the XML encrypted file in the following sequence:

1. Extract the encrypted content of the CypherValue element.
2. Read the algorithm attribute value of the EncryptionMethod element.
3. Read the Type attribute values of the EncryptedData element.
4. Obtain the keying information form the ds:KeyInfo element.
5. Use the information gathered in steps 1, 2, 3, and 4 to construct the plain text (decrypted) file.

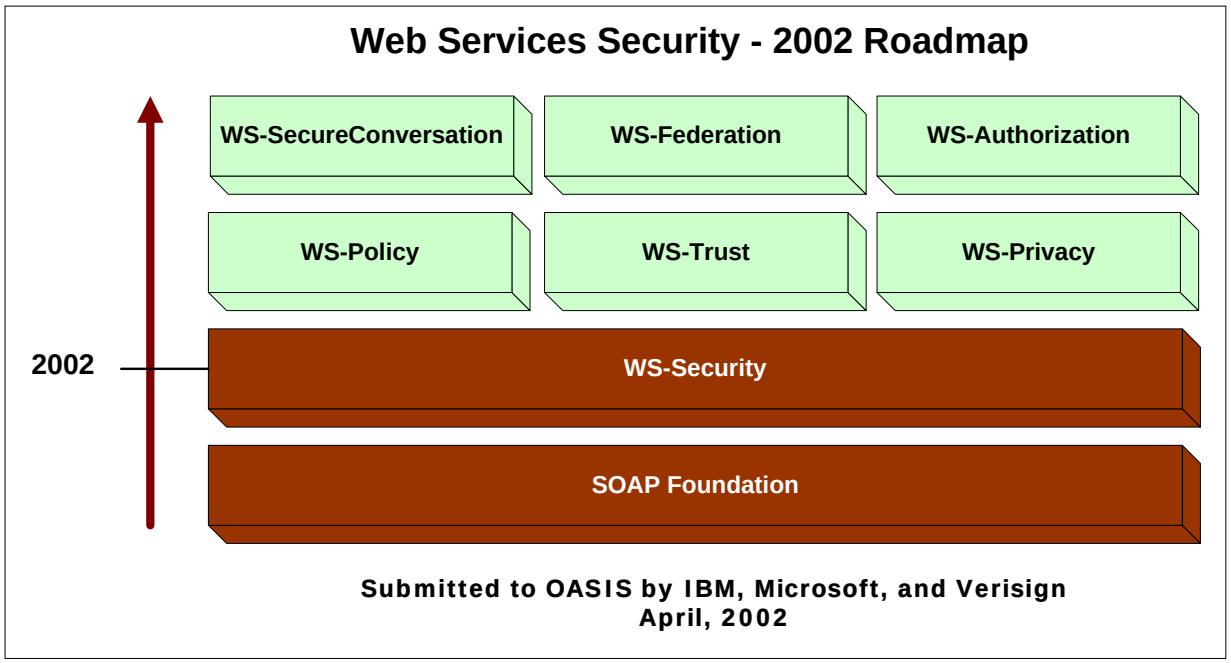
Web Services Security is Born

Microsoft introduces the notion of Web Services in 2001. They were a cornerstone in the .NET architecture vision.

IBM, Microsoft, and Verisign jointly released a security whitepaper in April 2002 title "*Security in a Web Services World: A Proposed Architecture and Roadmap*."
http://www-128.ibm.com/developerworks/library/specification/ws-secmap/

"This document describes a proposed strategy for addressing security within a Web service environment. It defines a comprehensive Web service security model that supports, integrates and unifies several popular security models, mechanisms, and technologies (including both symmetric and public key technologies) in a way that enables a variety of systems to securely interoperate in a platform- and language-neutral manner. It also describes a set of specifications and scenarios that show how these specifications might be used together"

This was a landmark document and the basis for the current standards today. It provided the following roadmap:

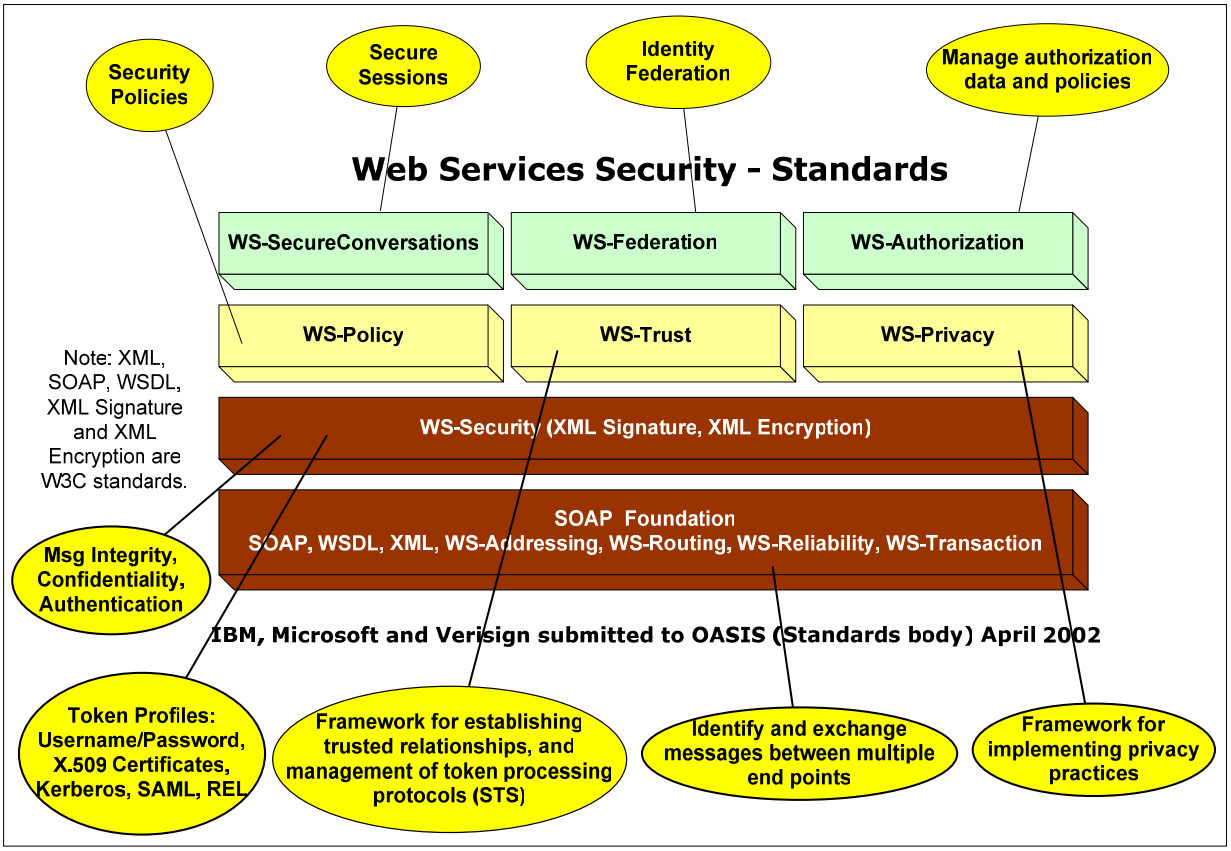


At the time of its release, there was agreement on SOAP Foundation and WS-Security. The upper layers were a vision at this point with details to be worked out. The paper was submitted to OASIS (Organization for the Advancement of Structured Information Standards). This included the notion of Security Tokens and defined Unsigned (Username), Signed (X.509 Certificates, SAML, and Kerberos Tickets). The paper also defined the vocabulary used in today's standards.

Web Service Standards and Architecture

In the past four years, many companies have joined the development of web security standards building on the 2002 base. In addition to completing the 2002 roadmap, many new WS* standards were determined along the way. Below is the Reference Security Architecture which has been annotated to briefly explain the purpose of each standard.

Reference Security Architecture

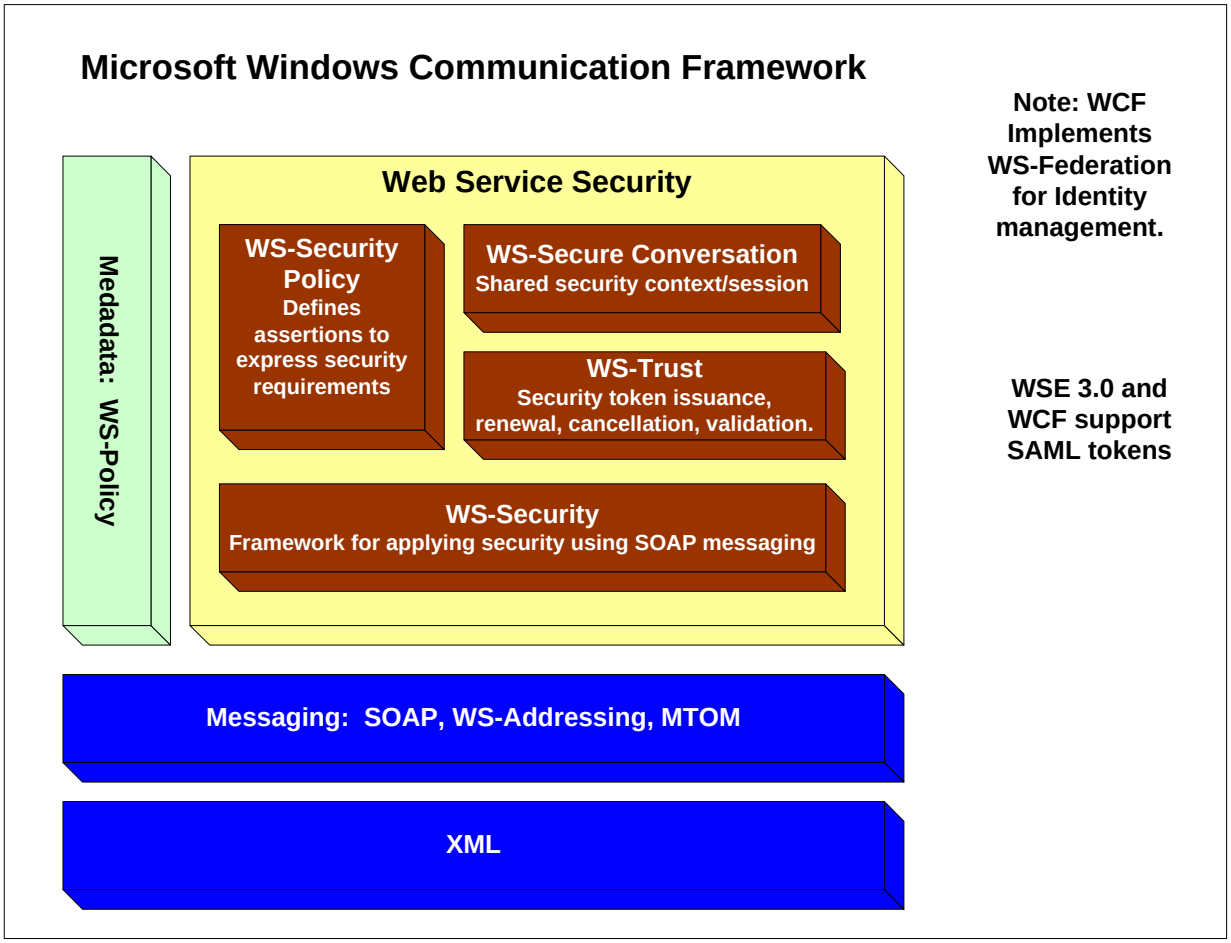


Example Vendor WS Security Architectures

Vendors implement the above standards in a manner that best positions them in a very competitive world. So, their diagrams may look a bit different, and not all vendors support all the standards. However, regardless of how vendors name their products and their components they should meet the same standards. This implies a high degree of interoperability among vendor web service products. In fact, this is a key principle of web services. The notion of location transparency, language and platform agnostic are the reason web services hold a lot of promise.

Microsoft is an avid supporter of web services and they have built native support into their .NET Framework, as well as their integrated development environment Visual Studio.

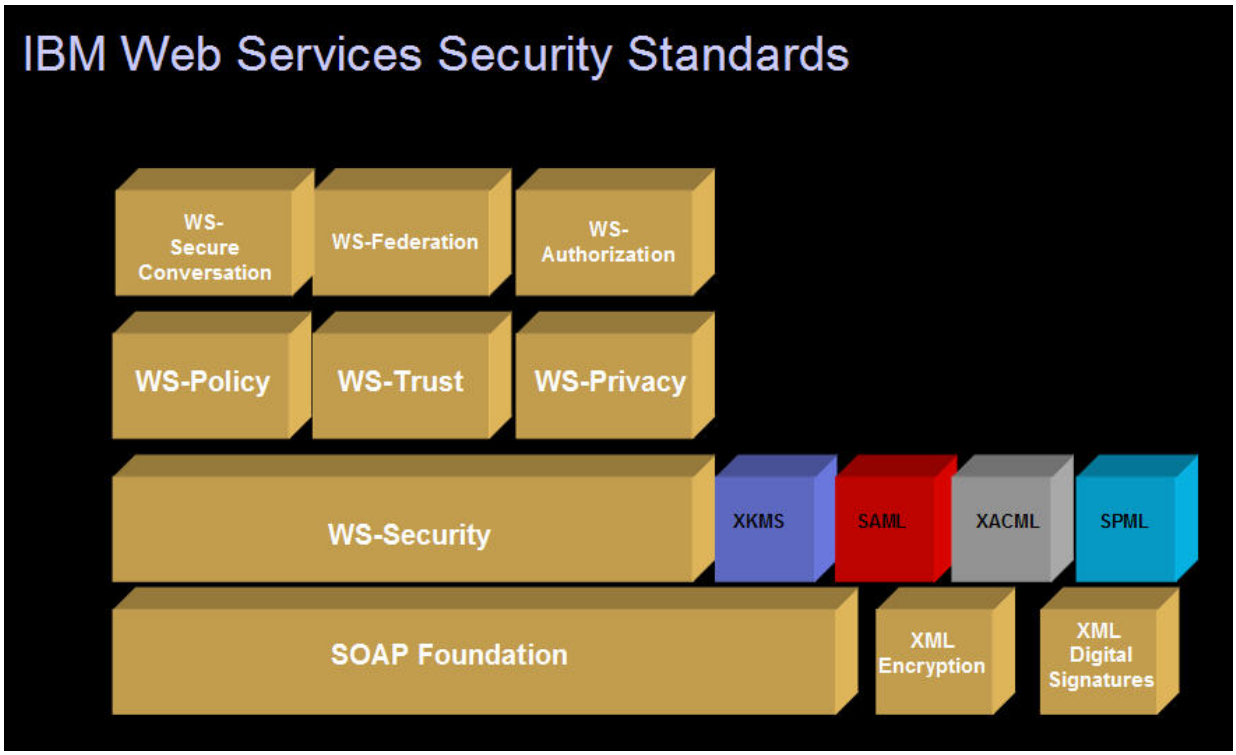
Below is the current WS* stack from Microsoft's perspective.



Note, this diagram is very similar to the Reference Security Architecture. Microsoft currently implements web services security in its Web Services Enhancements (WSE) product. WSE is not integrated with current production versions of Windows; however it is readily available via download and installs easily. WSE 3.0 added support for SAML tokens. Microsoft plans to implement the full suite of standards depicted in the above diagram in Windows Communication Framework which will be integrated with Windows Vista. This has a scheduled production date sometime in 2007.

IBM has also been a strong supporter of web services security and co-authored the original plan. They have built support into their current WebSphere line of products for the full stack. As an alternative, they also support the Liberty Alliance Framework and its underlying SAML-based architecture.

Following is the current WS* stack from IBM's perspective.



In additional to message authentication, the WS-Security standard defines message integrity, message confidentiality. The two standards that define integrity and confidentiality are XML Signatures and XML Encryption.

XML Signatures & Encryption Example

The Web Services Security (WSS) specification from OASIS defines the details of how to apply XML signature and XML encryption concepts in SOAP messaging. WSS relies on XMLDS and XML encryption for low level details and defines a higher-level syntax to wrap security information inside SOAP messages.

WSS describes a mechanism for securely exchanging SOAP messages. It provides the following three main security features:

1. Message Integrity
2. User Authentication
3. Confidentiality

It is an example SOAP message that carries security information according to the WSS syntax. Notice the request's header is carrying digital signature information.

```
<?xml version="1.0" encoding="utf-8"?>
<SOAP:Envelope
   xmlns:SOAP="http://www.w3.org/2001/12/soap-envelope"
   xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext"
   xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
   <SOAP:Header>
      <wsse:Security>
```

```
        <wsse:BinarySecurityToken
           ValueType="wsse:X509v3"
           EncodingType="wsse:Base64Binary"
           wsu:Id="MyCertificate">
             LKSAJDFLKASJDlkjlkj243kj;lkjLKJ...
        </wsse:BinarySecurityToken>
        <ds:Signature>
           <ds:SignedInfo>
              <ds:CanonicalizationMethod
                 Algorithm="http://www.w3.org/2001/10/xml -exc-c14n# "/>
              <ds:SignatureMethod
                 Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
              <ds:Reference URI="#myRequestBody">
                 <ds:Transforms>
                    <ds:Transform
                       Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                 </ds:Transforms>
                 <ds:DigestMethod
                    Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                 <ds:DigestValue>BSDFHJYK21f...</ds:DigestValue>
              </ds:Reference>
           </ds:SignedInfo>
           <ds:SignatureValue>
              GKLKAJFLASKJ52kjKJKLJ345KKKJ...
           </ds:SignatureValue>
           <ds:KeyInfo>
              <wsse:SecurityTokenReference>
                 <wsse:Reference URI="#MyCertificate"/>
              </wsse:SecurityTokenReference>
           </ds:KeyInfo>
        </ds:Signature>
     </wsse:Security>
  </SOAP:Header>
  <SOAP-ENV:Body>
     <s:GetMyAccountBalances
        xmlns:s="http://ftb.ca.gov/partnerservice/"
        ID="myRequestBody">
           <!--Parameters passed with the method call-->
     </s:GetMyAccountBalances>
  </SOAP-ENV:Body>
```

Here are the simple points about the above listing that will help you understand WSS syntax:

1. The `SOAP:Envelope` element contains namespace declarations for SOAP, WSS, and XMLDS.

2. The `SOAP:Header` element contains just one child element (`wsse:Security`), which is the wrapper for all the security information. The `wsse:Security` element has two child elements, namely a `wsse:BinarySecurityToken` element and a `ds:Signature` element.

3. The `wsse:BinarySecurityToken` element contains a security token. A security token is like a security pass or an identity card that you are required to show if you want to enter a restricted access area. There are several types of electronic security tokens.

   The most popular and widely used security token is a login-password pair, like the one you use while checking your e-mail.

   A login-password pair is a human readable security token. There are some security tokens that are in binary form (and therefore not necessarily human readable). Such tokens are referred to as binary security tokens. For example an X509 certificate (a widely popular format for digital certificates developed by ITU-T) is a binary security token.

   The `ValueType` attribute of the `wsse:BinarySecurityToken` element tells what type of binary security token is wrapped inside this `BinarySecurityToken` element. The `ValueType` attribute contains `wsse:X509v3` as its value, which identifies X509 certificates.

   The `EncodingType` attribute of the `wsse:BinarySecurityToken` element tells the encoding of the binary security token. As already explained, it is not possible to wrap binary data inside XML format as such. Therefore, we have to encode binary data (usually as a sequence of base-64 encoded values) before wrapping inside XML. The X509 certificate is wrapped inside the `wsse:BinarySecurityToken` element as the element content.

4. The `ds:Signature` element is the same as the one we discussed in the section on XML signatures. Note two important things:

   - Look at the `URI` attribute of the `Reference` element. Its value (`#myRequestBody`) is a fragment identifier that points toward the SOAP:Body element. This means that the SOAP:Body element is the one that we have signed and wrapped the signature in XMLDS tags.

   - Secondly, also look at what the `ds:KeyInfo` element contains. It is a `wsse:SecurityTokenReference` element. The `wsse:SecurityTokenReference` element contains references to security tokens. In our case, it has a child element named `wsse:Reference`, whose URI attribute points toward the `wsse:BinarySecurityToken` element discussed in point 3 above. This means that the public key inside the X509 certificate (which the `wsse:BinarySecurityToken` element wraps) will be used to verify the signature.

Federated Identity and Authentication

Federated Identity Management

Authentication means verifying the identity of a user. When you check your e-mail, you enter your username and password to get authenticated. It is assumed that you have kept your password confidential. Therefore the knowledge of your password is used to make sure that you are the one who is trying to check your email. This is a weak form of authentication.

Similarly, one can use a stronger form of authentication such as certificates. An X509 certificate can be wrapped within the SOAP header. The certificate is actually a security token (just like a password) that the recipient of the WSS message can use in order to authenticate the user before allowing access to a web service.

In additional to X.509 Certificates, the WS-Security standard supports other types of strong authentication, such as SAML and Kerberos. SAML (Security Access Markup Language) is an OASIS standard that is supported by both WS-Federation and Liberty Alliance. Kerberos is supported by WS-Federation, but not Liberty.

In a federated identity environment, an entity may be associated with more than one identity provider. The mechanisms employed by each provider may be of different strengths and some application contexts may require a minimum to accept the claim to a given identity.

For example, this means a person could enter *any* application in a trusted domain with a specific identity assertion. The identity providers associated with this trusted domain will sort out whether or not this particular identity can be authenticated and to what level.

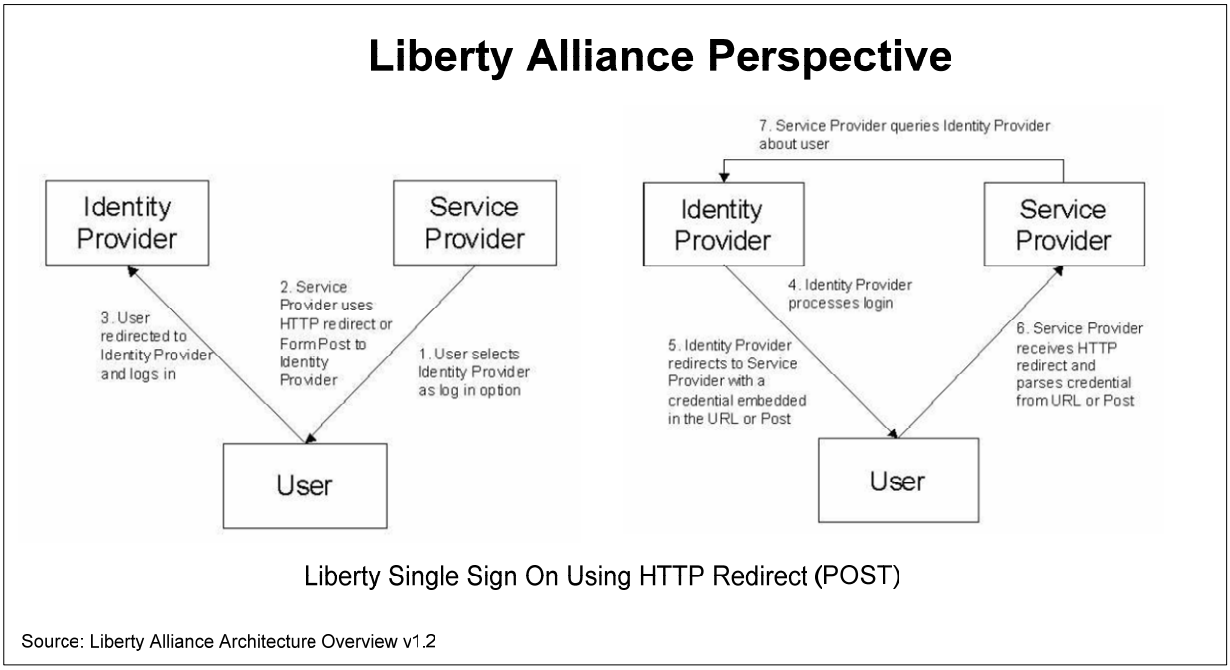
Web Services Federation (WS-Federation)

WS-Federation was first defined in July 2003 by IBM, Microsoft, BEA, Verisign, and RSA Security. WS-Federation is part of an overall effort to build a web services security framework. The specification includes WS-Language which defines how different security realms broker identities. It also includes Passive Requestor Profile which describe how federation helps provide identity services to HTTP 1.1-based browsers, Web-enabled cell phones and devices; and Active Requestor Profile which does the same for applications based on SOAP and other smart clients.

Liberty Alliance

The Liberty Alliance, a consortium representing organizations from around the world, was created in 2001 to address the technical, business, and policy challenges around identity and identity-based Web services. (http://www.projectliberty.org/index.php). Today, it has more than 150 member companies including IBM, Sun, SAP, Oracle, BEA, and HP.

Liberty Alliance produced an Identity Framework. It is based on SAML (Security Access Markup Language) – another OASIS standard. SAML is a security token framework. Microsoft, who is not a member of Liberty, recently provided support for SAML 2.0 at the token profile level. However, Microsoft uses the WS-Federation framework instead of the Liberty framework.



Microsoft

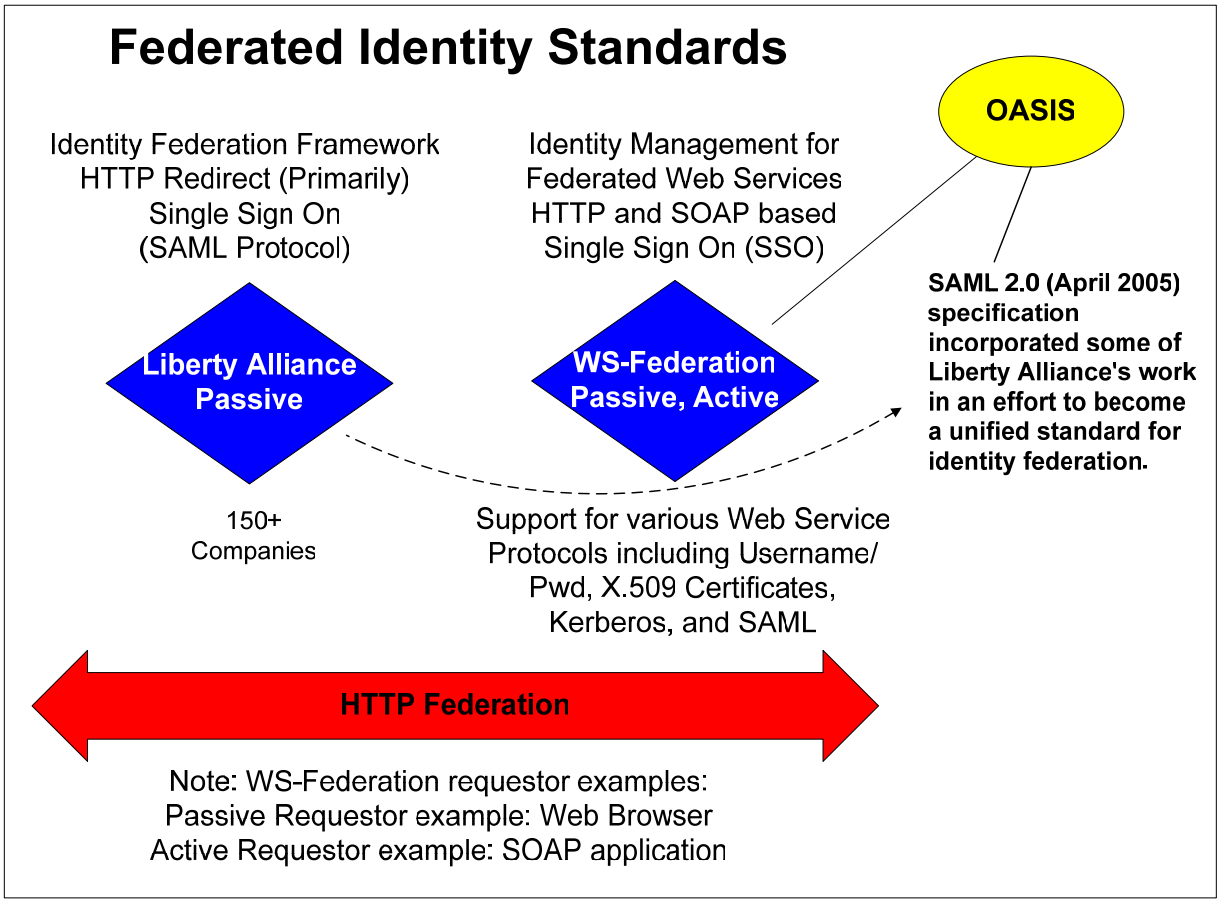
Microsoft originally created .NET Passport to handle network identity. This product was renamed to Passport Network. Recently, it was re-branded to Windows Live ID and is used by consumers and participating merchants that want a seamless authentication experience while avoiding managing credentials.

WSE (Web Services Extensions) Toolkit was first released December, 2002 and supported WS-Security. WSE 2.0 was released July 2003 and added a message-based object model and additional support for WS-Security, WS-Policy, WS-SecurityPolicy, WS-Trust, WS-SecureConversation, and WS-Addressing. WSE 3.0 was released November, 2005 and added support for MTOM (Message Transport Optimization Mechanism), and SAML Secure Token Service (STS). Note that WSE is a separate download and an add-on to the current Windows platform.

WCF (Windows Communications Framework), code named Indigo, is the future. WCF will be integrated with Windows Vista (the next Windows platform) due in 2007. Among other enhancements, WCF will provide support for federated identity via WS-Federation.

Federated Identity Standards

The primary method for achieving federated identity management has been via Liberty Alliance and SAML. The Liberty Identity Framework (ID-FF) employs an architecture known as HTTP Redirect. ID-FF is based on the SAML protocol (an OASIS standard) to handle credentials via a “security token”. Since ID-FF is primarily Web browser-based, it is classified passive (Passive Requestor Profile).



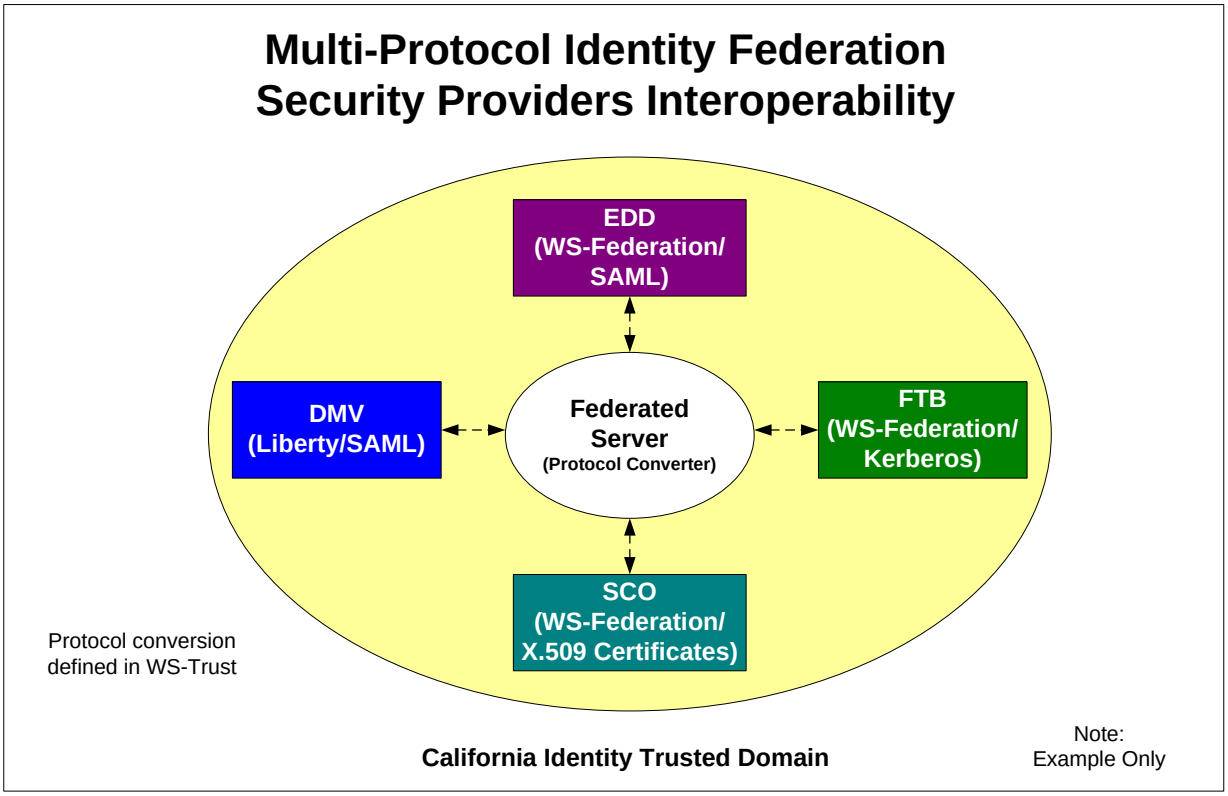
In contrast, WS-Federation (an OASIS standard), also a federated identity framework but broader based. It supports not only SAML, but also Kerberos, Username/Password, and X.509 Certificates. It is anticipated that at some point Liberty Alliance and WS-Federation will merge into one standard for federated identity management.

Federated Identity Interoperability

One of the key reasons for federated identity is Web Single-Sign On (SSO). Users typically have to log onto multiple systems and remember multiple passwords. This is also a security problem because sensitive user information is maintained in multiple locations, often containing different information

The idea of SSO is for users to log on once, and then their identity is "trusted" by other applications that participate in the trusted relationship. Liberty Alliance introduced this concept with the notion of "identity providers". Regardless of how users get to your website, they will be redirected to an appropriate identity provider to be authenticated. Once they have been successfully authenticated, they will be returned to your site with a set of "credentials" –often in the form of a SAML message.

Theoretically, different departments within California could employ different identity frameworks and tokens which would still interoperate as long as they adhered to WS-Security standards and Liberty Alliance. For example, the following diagram shows mixing Liberty and WS-Security frameworks as well as various token types. Protocol conversion is handled by a Federated Server.



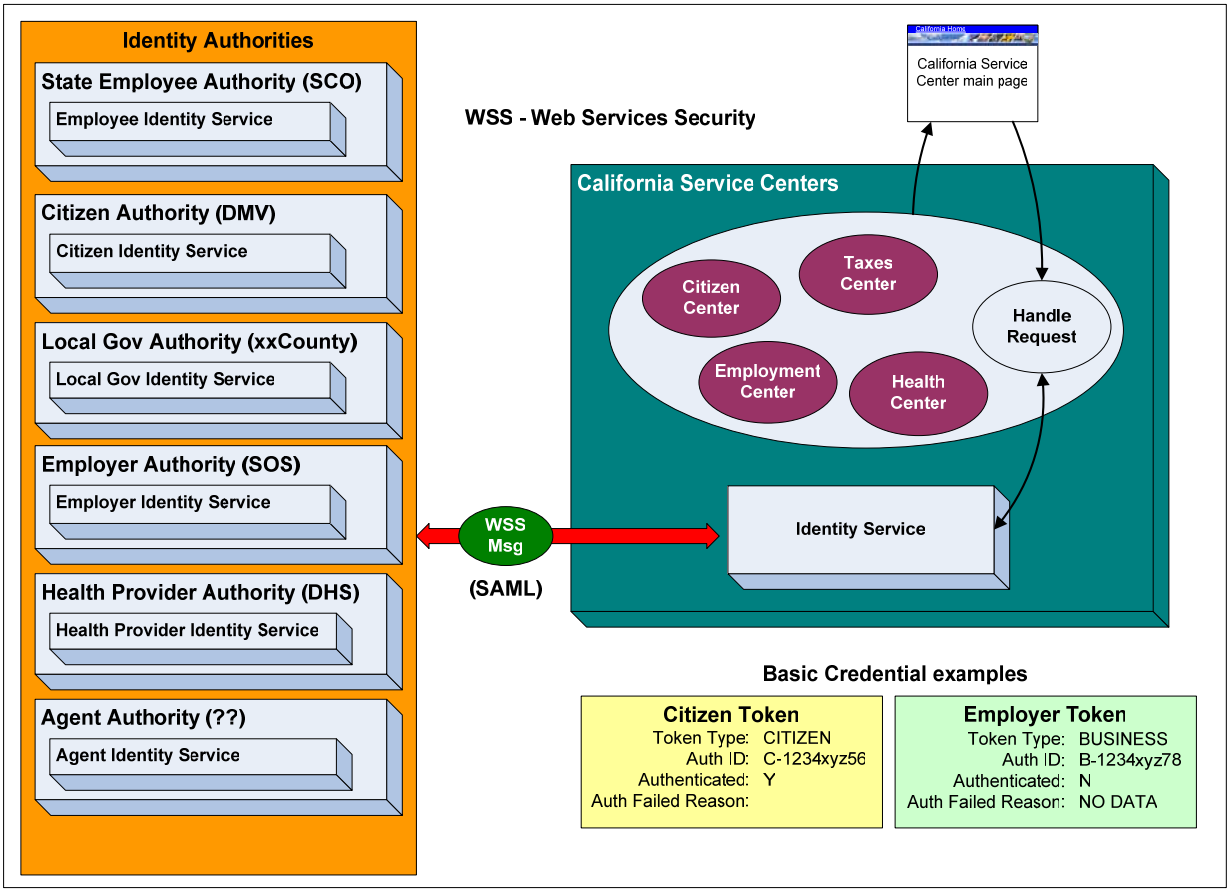
While this works it makes more sense to choose a single token definition such as SAML. Eventually, Liberty Alliance and WS-Federation will merge, so either (or a combination of them) could be used for the framework.

Federated Identity Management Example

To simplify identity management, a single department could be designated to handle identities of the same type. For example, State Controllers Office might handle State employees, Department of Motor Vehicles citizens, and Los Angeles County local government.

Businesses are a bit more complicated. It is likely that multiple identity providers will be needed to handle the diverse nature of business types. For example, Secretary of State might the authority for employers, and Department of Health Services might handle health care providers.

An identity service might be running at a centralized location (California Service Center in the following example) which would determine if identity must be established based on the selected user interaction. This service would then invoke the appropriate identity authority and return the appropriate basic credentials, disapproval code, or an error.

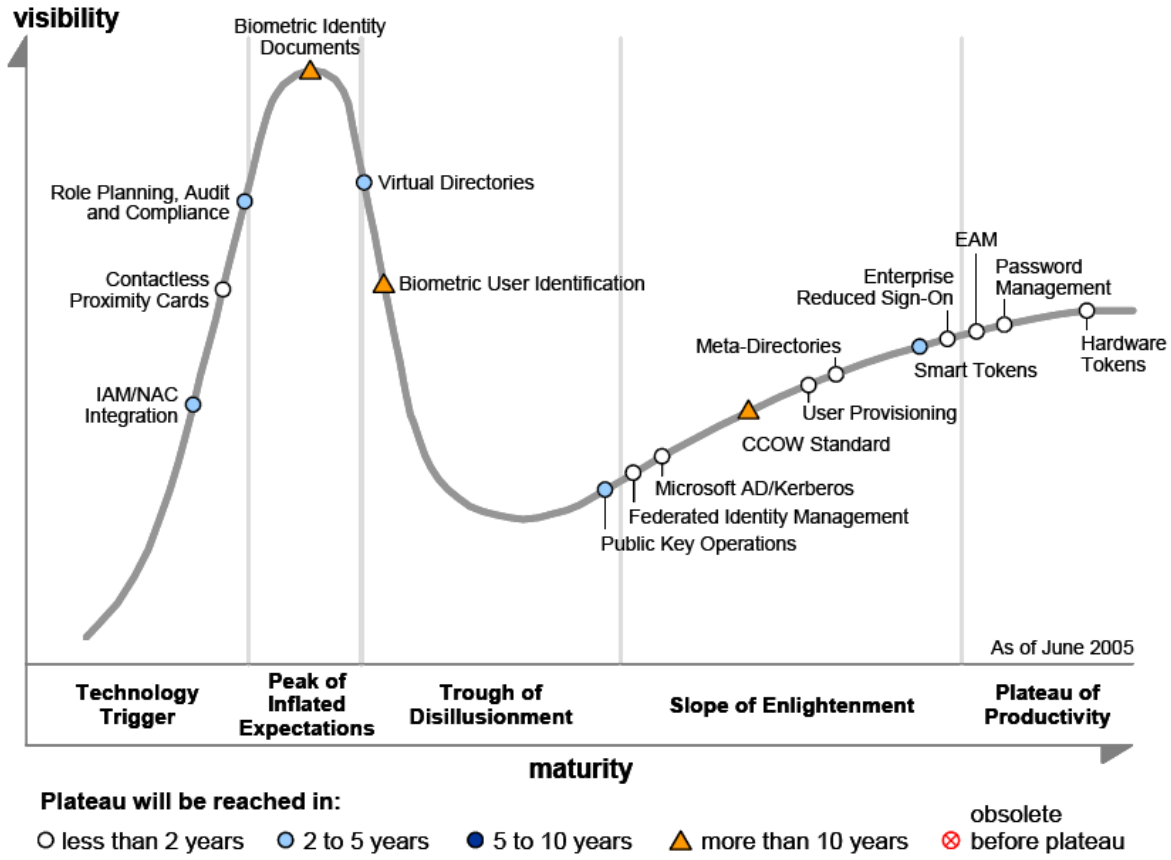


Identity Authorities Pattern

Gartner's Hype Cycle for Identity and Access Management

Most people are aware of Gartner's "Magic Quadrants" and vendors strive to position themselves in the upper right hand quadrant. However, Gartner also provides a series of "Hype Cycles" which track topics along a maturity curve. The Hype Cycle for Identity and Access Management Technologies shows Federated Identity Management, User Provisioning, and Reduced Sign-On as all being in the "Slope of Enlightenment" area which means they are fairly mature and people are successfully using these technology components.

Figure 1. Hype Cycle for Identity and Access Management Technologies, 2005



Security Access Markup Language (SAML)

Introduction to SAML

SAML is an XML vocabulary that defines the syntax necessary to exchange identity information between applications. The identity information is exchanged in the form of assertions. A security provider service is responsible for providing assertions about its trusted partners and therefore acts as an SAML assertion authority.

For example, the California Service Center (CSC) might request an assertion from the Secretary of State Business Provider Service (a *SAML authority*). CSC is a *requester application* and the *subject* of the assertion as well. After getting the assertion from the provider service, CSC will wrap the assertion in a WSS message and send the WSS message to the appropriate department application. The receiving department will rely on the assertion to decide whether to allow access to its application. The receiving department is a *relying party*. Notice that the SAML specification does not define any security attributes by itself. SAML users are expected to design their own security attribute namespaces.

SAML Example

In the following listing, we have wrapped an SAML assertion in a WSS message.

In order to understand what information this listing contains, you need to compare it with the previous listing in #An Introduction to Web Service Security. There are some differences between the two:

1. There is no `BinarySecurityToken` element in the following listing. Instead of a security token, we have an assertion. The `Assertion` element appears as a child of the `wsse:Security` element, just like the `BinarySecurityToken` element in the first listing.

2. There are two `ds:Signature` elements in the following listing. The first appears within the `Assertion` element. The SAML authority produced this signature while issuing the assertion, so that any application who receives this assertion can verify its integrity. We have not shown the details of this signature for the sake of simplicity.

   The second `ds:Signature` appears as a direct child of the `wsse:Security` element. This signature is from our SAML authority, which produced the signature over the `GetMyAccountBalances` element in the SOAP body while authoring the request. Compare the `ds:Signature` element in previous listing with the `ds:Signature` element in the following listing. Both these `Signature` elements were produced by the application. The one difference is their `ds:KeyInfo` elements.

   In the previous listing, the `ds:KeyInfo` element referred to the certificate wrapped inside the `BinarySecurityToken` element. But in this the following listing, there is no `BinarySecurityToken` element. Instead, we have an `Assertion` element acting as a security token. Therefore it makes sense to refer to the assertion from the `ds:KeyInfo` element.

3. As already explained, the `ds:KeyInfo` element in the following listing refers to the assertion. When the message reaches the relying party, they will need to validate the signature in order to verify requester's identity as well as the integrity of the message. Therefore the recipient will need a public key to verify the signature. Where is the public key that the application can use to verify the signature? The `Assertion` element is the most relevant place to look for the public key.

   There is only one key inside the `Assertion` element. Its name is "MyKey". The application will use this key to verify the signature.

4. Notice the `SubjectConfirmation` element within the `Assertion` element, which specifies the relationship between the subject and the author of the message that contains the assertion.

   The `SubjectConfirmation` element should specify that the subject authored the message that contains this assertion. The `SubjectConfirmation` element has two child elements, namely a `SubjectConfirmation` and a `ds:KeyInfo` element. The two child elements form a pair.

   The `ConfirmationMethod` element wraps the string identifier for the holder-of-key method that we discussed earlier. The holder-of-key method simply specifies that the author of this message is the subject of the assertion and it holds the key wrapped by the accompanying `ds:KeyInfo` element. Notice that the accompanying `ds:KeyInfo` element, which is a sibling of the `ConfirmationMethod` element, wraps the key named "MyKey"

   I have already said that the tour operator uses the same key (named MyKey) to sign the `GetMyAccountBalances` element. This provides a link between the WSS message author and the subject of the assertion. The application will simply need to verify the integrity of the assertion (by verifying the signature of the SAML authority) and the signature of the requesting application. If the two signatures validate, the recipient application can be sure that the assertion is not fake and it is really asserting the author of the WSS message.

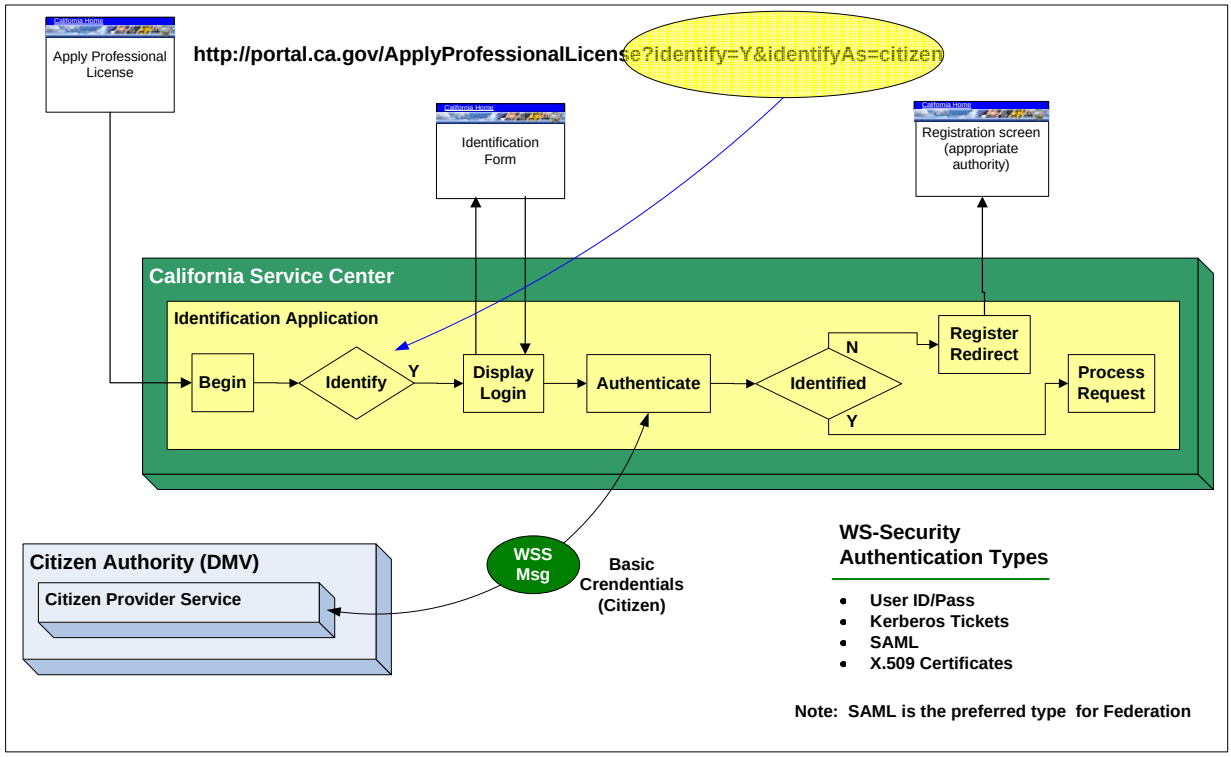
```
<?xml version="1.0" encoding="utf-8"?>
<SOAP:Envelope
   xmlns:SOAP="http://www.w3.org/2001/12/soap-envelope"
   xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/xx/secext"
   xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
   <SOAP:Header>
      <wsse:Security>
         <saml:Assertion
            xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
            MajorVersion="1"
            MinorVersion="0"
            AssertionID="http:// ftb.ca.gov/AuthenticationService/SAMLAssertions/786"
            Issuer="http:// ftb.ca.gov"
            IssueInstant="2003-03-11T02:00:00.173Z">
            <Conditions
               NotBefore="2003-03-11T02:00:00.173Z"
               NotOnOrAfter="2003-03-12T02:00:00.173Z"/>
            <AttributeStatement>
               <Subject>
```

```
            <NameIdentifier
              NameQualifier=”http:// ftb.ca.gov”>
              MyTourOperator
            </NameIdentifier>
            <SubjectConfirmation>
              <ConfirmationMethod>
                urn:oasis:names:tc:SAML:1.0:cm:holder-of-key
              </ConfirmationMethod>
              <ds:KeyInfo>
                <ds:KeyName>MyKey</ds:KeyName>
                <ds:KeyValue> ... </ds:KeyValue>
              </ds:KeyInfo>
            </SubjectConfirmation>
          </Subject>
          <Attribute
            AttributeName=”CitizenStatus”
            AttributeNamespace=”http:// ftb.ca.gov /AttributeService”>
            <AttributeValue>TaxLevel5</AttributeValue>
          </Attribute>
        </AttributeStatement>
        <ds:Signature>...</ds:Signature>
      </saml:Assertion>
      <ds:Signature>
        <ds:SignedInfo>
          <ds:CanonicalizationMethod
            Algorithm="http://www.w3.org/2001/10/xml -exc-c14n# "/>
          <ds:SignatureMethod
            Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
          <ds:Reference URI="#myRequestBody">
            <ds:Transforms>
              <ds:Transform
                Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
            </ds:Transforms>
            <ds:DigestMethod
              Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <ds:DigestValue>BSDFHJYK21f...</ds:DigestValue>
          </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>
          GKLKAJFLASKJ52kjKJKLJ345KKKJ...
        </ds:SignatureValue>
        <ds:KeyInfo>
          <wsse:SecurityTokenReference
            <wsse:KeyIdentifier wsu:id="SAML786Identifier"
              ValueType=”saml:Assertion”>
              http:// ftb.ca.gov /AuthenticationService/SAMLAssertions/786
            </wsse:KeyIdentifier>
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
      </ds:Signature>
    </wsse:Security>
  </SOAP:Header>
  <SOAP-ENV:Body>
    <s: MyAccountBalances
      xmlns:s=“http:// ftb.ca.gov/partnerservice /”
```

```
        ID="myDiscountRequestBody">
            <!--Parameters passed with the method call-->
        </s: MyAccountBalances>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

A Citizen Request Example

Let's illustrate the above security principles in an example scenario. In this case, a citizen will apply for a professional license (doctor, dentist, real estate, CPA, etc.). They will first go to the new California Service Center which will have a link (or picture, or button) they can clink on to start the online application process.

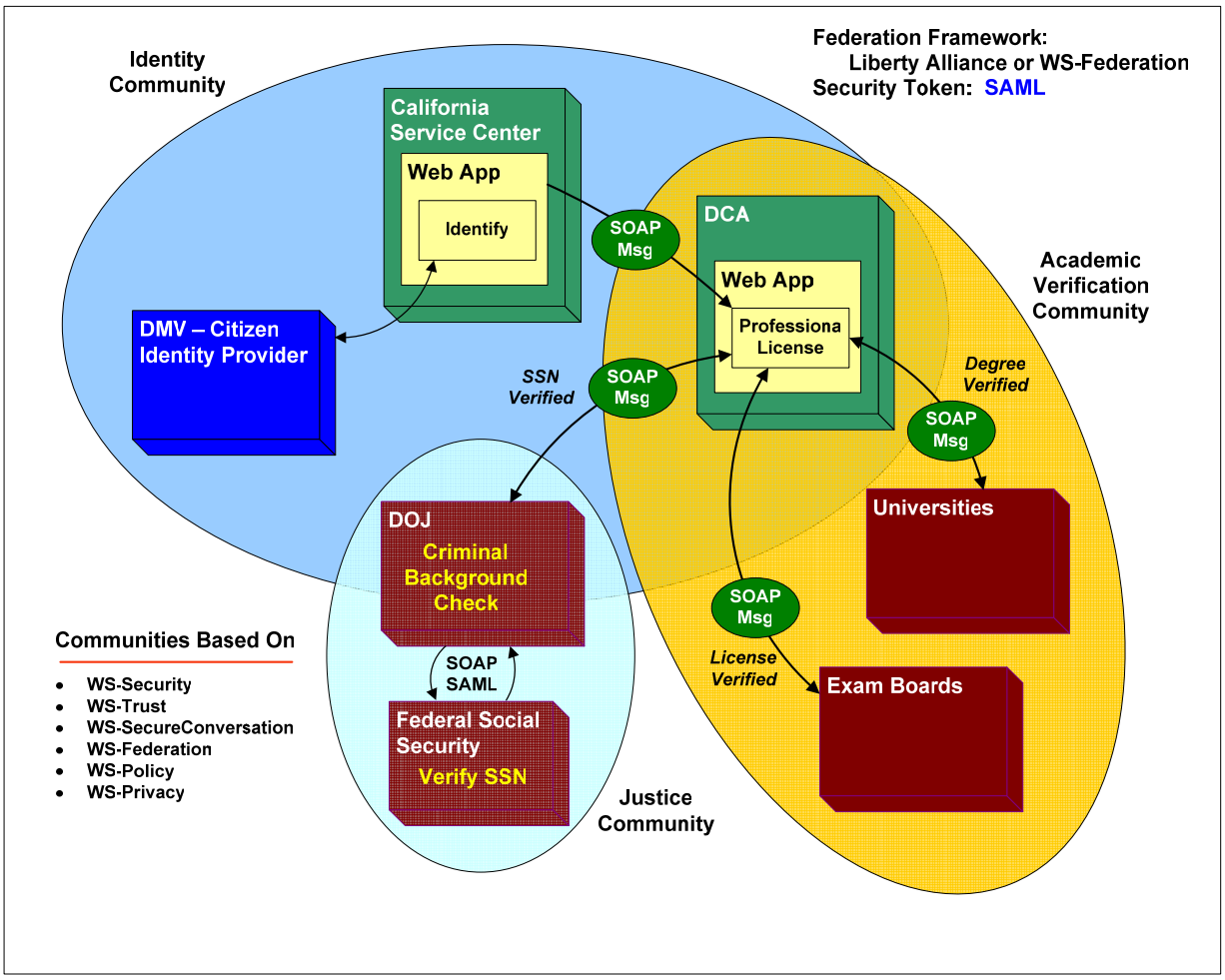


Notice, the onclick action is encoded with two parameters: identify=y and identifyAs=citizen. The CSC web application will check these parameters and invoke the Identity process if required. In this case, because we want to be identified as a Citizen, a request will be sent to the DMV Citizen Provider Service – which, for the purpose of this illustration, we have designated DMV as the Authority for identifying citizens within the State.

In reality, one would not embed the above parameters in the URL since they would appear as clear text on the browser status line. Rather, they would be hidden form variables and sent via the HTML POST method.

The Citizen Provider Service will return a WSS message (Web Services Security message) containing the basic credentials for this person. It will be up to the appropriate architects with the State to define the exact details of basic credentials. The CSC web application will then redirect the request to the appropriate department application to handle the actual professional license application process.

Let's say CSC invoked a Professional License web application managed by the Department of Consumer Affairs (DCA). The DCA application would look at the SOAP message and then send a request to the DMV Citizen Provider Service. It would receive a return message containing the authentication status and basic credentials.



Communities of Interest Pattern

The DCA Professional License application would then consume services at a University to verify that this person had the appropriate degree. It would also consume a service at the appropriate Exam Board to verify that this person had a proper license. Additionally, it would consume the Criminal Background Check service at DOJ, which might send a message to the Federal Social Security Administration Verify SSN service to confirm the SSN.

Notice there are three circles of trusts. In the above example because DMV, DTS, DCA and DOJ are part of the Identity Circle of Trust, DCA and DOJ will use the identity information initiated by DTS and provided by DMV. Further, because Universities and Exam Boards and in the Academic Verification Circle of Trust, they will use the identity information passed to them by DCA.

However, if DOJ were to invoke a service at an Exam Board that required identification, it would not be honored because they are not part of the same circle of trust. So, re-identification would be required.

Also keep in mind that different types and levels of security can be used. That is, one might use certificates, PKI, Kerberos, or other types of tokens. They are all part of the WSS standards.

User Provisioning

This section will be detailed in a future release.

SOA Firewalls for Web Security

Use of SOAP and XML can expose new risks to your organization that could potentially let intruders penetrate core business services. Packet-level firewalls can't help you secure Web services traffic because they can't detect SOAP and XML traffic. For example, because SOAP typically uses HTTP or SMTP, it easily passes through traditional firewalls—a phenomenon known as the port 80 problem. Therefore, a new kind of “firewall” has appeared: the Edge Enforcement Agent, which can enforce security policy embedded inside a SOAP message.

Like HTML, XML is a markup language that provides a platform-independent standard for exchanging information between systems on the intranet and Internet. HTML differs from XML, however. HTML is static: It provides a finite set of ways to structure text information. When new needs arise, the HTML standard must be updated to accommodate them. In contrast, XML is a more abstract markup language that provides built-in extensibility through a schema that you define.

Using XML provides a way to format or structure data and commands or transaction requests. Two applications that support the same XML schema can easily exchange data and request transactions. But although XML lets you assemble a message, it doesn't address getting the message from the client to the server and back again. That task is the job of a protocol—SOAP, in the case of Web services.

SOAP gives applications a way to send XML-based messages over a network within HTTP or SMTP. When one application needs another application's services, the first application formats a service request (i.e., a function name and parameters) into XML, then packages the request in a SOAP envelope and sends it. The target application opens the envelope, executes the request, then uses SOAP to return a response. Environments such as Windows .NET Framework let the application developer work at a high level of abstraction, but the Framework still relies heavily on SOAP and XML, so related security concerns still come into play.

Because of XML's platform-independent nature and its ability to let disparate systems interface easily, most Web services use well-known XML schemas and consequently are vulnerable to a much broader variety of potential attacks than are narrower technologies such as Distributed COM (DCOM) and EDI. As a result, you face a greater likelihood of people sniffing the data, non-authenticated clients directly connecting to and trying to retrieve data from your Web services server, and Denial of Service (DoS) attacks that use malformed messages to exploit a well-known schema.

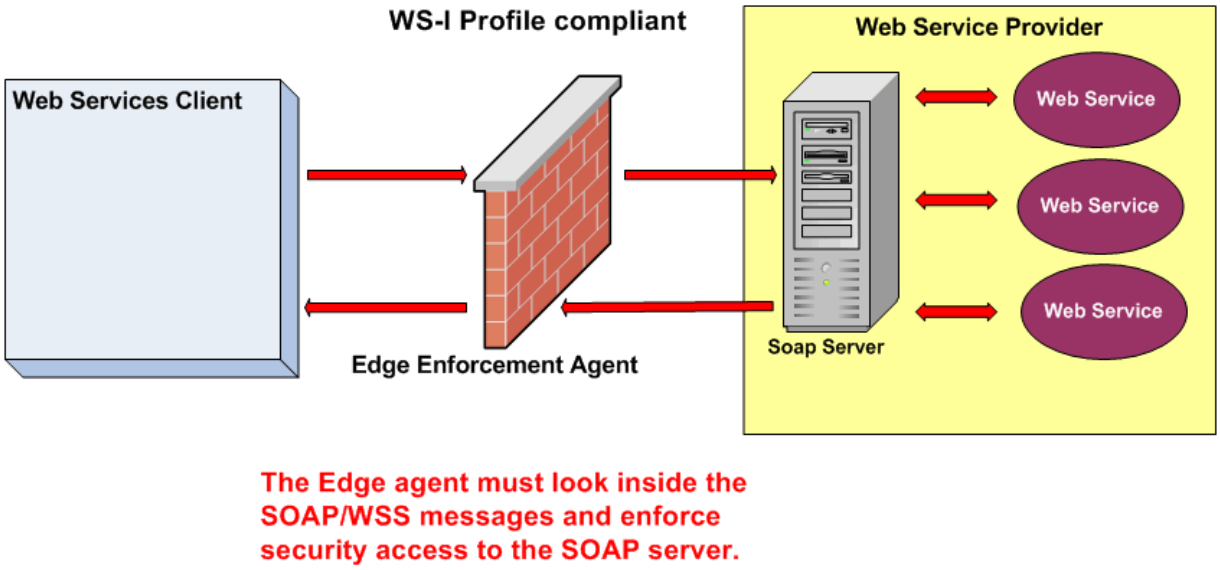
Traditional firewalls, which look at the world in terms of IP addresses, ports, and protocols, address risks that occur at a much lower level than the level at which SOAP and XML reside. Instead of determining whether to pass a given packet to the internal network, SOA firewalls validate traffic in terms of Web services, individual messages, and data elements and evaluate whether to let a given requester access a specific operation. XML-embedded malware, such as worms, Trojan horses, and DoS attacks, are risks with SOAP and XML.

You can address SOAP/XML security concerns three ways. First, if your use of SOAP/XML is light and limited to a stable set of partners, you might be able to get by with a classic firewall. However, the vendor must enhance the firewall so that it can at least recognize SOAP within HTTP and other protocols. You can then enable SOAP and XML content between your organization and its trusted business partners and block everything else.

A second option for SOAP/XML firewalls is to build your own. Although probably not an appealing alternative for most organizations, building your own firewall is possible, and tools exist to help you do

the job. For example, Microsoft Internet Security and Acceleration (ISA) Server 2000 lets you write Internet Server API (ISAPI) filters on an ISA server, and Microsoft provides a model ISAPI filter for validating SOAP/XML messages while they're at the ISA server.

The third, and usually best, option is an application-level SOA firewall that operates behind your classic firewall to validate only SOAP/XML traffic. Similar to a proxy, this type of product receives the Web service message as though the application-level firewall were actually the Web service. These products inspect the message; authenticate the person, program, or organization that sent it; then verify that the sender is authorized to the Web service and the requested operation. Authentication can use a simple username and password, a certificate, or a federated system that uses Security Assertion Markup Language (SAML).



An Edge Enforcement Agent can authenticate credentials against sources such as a Lightweight Directory Access Protocol (LDAP) directory (e.g., Active Directory—AD) or a Remote Authentication Dial-In User Service (RADIUS) server. Then, the agent checks the requested Web service and operation and the data elements (i.e., parameters) within the message to make sure the request is valid and authorized for the user. Either before or after authentication, depending on the product, the agent weeds out malformed messages and DoS attacks by ensuring that the request's format complies with the corresponding schema. The agent forwards messages that pass these checks to the appropriate Web service.

Most agents also provide some type of audit functionality and logging so that you can monitor what's happening with your Web services. Because encryption and XML parsing are CPU-intensive, this more complex proxy architecture is important to implementing SOA firewalls in high-security and high-volume Web service scenarios. Because SOAP/XML supports security at the transport level, a SOA firewall can use Secure Sockets Layer (SSL) and Transport Layer Security (TLS) to encrypt the entire HTTP-based message stream.

But sometimes you need to be able to encrypt or digitally sign portions of an XML document—to facilitate multiparty transactions, for example. The XML Encryption and XML Signature security standards meet these intra-document cryptography needs. Because a SOA firewall functions as a proxy

Web service, all authentication, encryption, and decryption take place at the firewall, letting you centrally and consistently control authentication, encryption, and policy checks even if Web services are scattered on servers throughout your network. Another advantage is that, because only decrypted traffic can be inspected, encrypted content is decrypted at the firewall and compared against the firewall's policy.

Also keep in mind that an Enterprise Service Bus can enforce message security. So, one might use a combination of XML-aware firewalls and an ESB.

Security Standards for Web Services

Web services are still evolving and as a result there are a large number of standards. Here is a quick list of some of the more important security standards related to web services. There are many more standards, so for a more complete lists see Web Services White Paper "*Web Service Standards*" section.

Standards Organizations:

W3C - World Wide Web Consortium http://www.w3.org/

OASIS – Organization for the Advancement of Structured Information Standards http://www.oasis-open.org/home/index.php

WS-I – Web Services Interoperability Organization - Provides interoperability standards in the form of Profiles. http://www.ws-i.org/ Current profiles include:

- Basic Profile (V1.0, V1.1, Simple SOAP Binding Profile 1.0)
- Attachments Profile 1.0
- Basic Security Profile (V1.0, Security Scenarios)

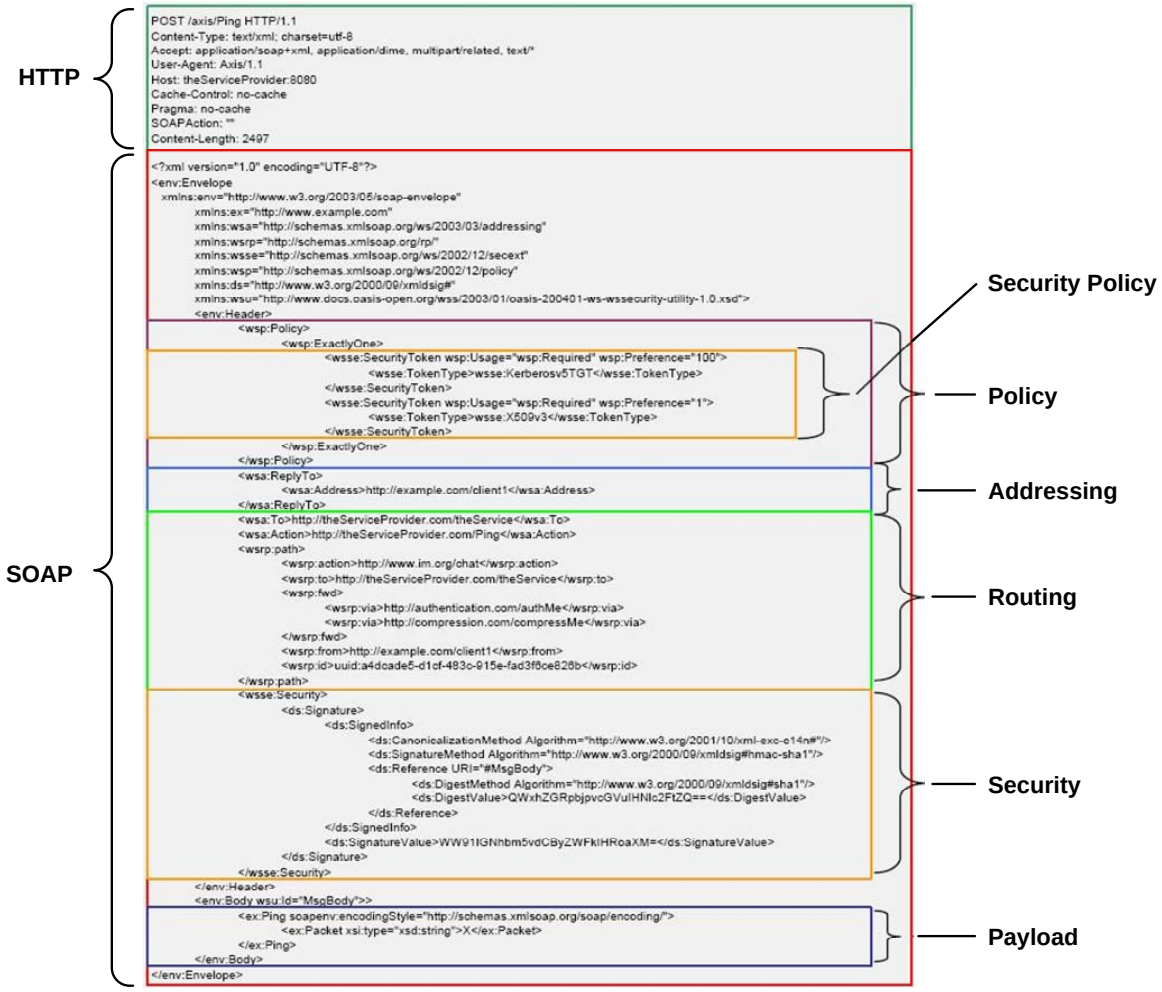
Liberty Alliance - http://www.projectliberty.org/

Security Standards:

WS-Security http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss
WS-Trust/Secure Token Service http://www-128.ibm.com/developerworks/library/specification/ws-trust/
WS-Provisioning http://www-128.ibm.com/developerworks/library/specification/ws-provis/
WS-Federation http://www-128.ibm.com/developerworks/library/specification/ws-fed/
WS-Authorization
http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp?topic=/com.ibm.websphere.express.doc/info/exp/ae/cwbs_wssv6chron.html

WS-Policy http://www-128.ibm.com/developerworks/library/specification/ws-polfram/
WS-Privacy http://www.webserviceshelp.org/wsh/Standards/Web+Services+Security/WS-Privacy.htm
SAML (Secure Access Markup Language) http://en.wikipedia.org/wiki/SAML
WS-SecureConversations http://www-128.ibm.com/developerworks/library/specification/ws-secon/

Web Services Message Stack Example



Web Services Message with Security Example